Counting Database Repairs under Primary Keys Revisited

Counting Database Repairs under Primary Keys Revisited

Marco Calautti
School of Informatics
University of Edinburgh
mcalautt@inf.ed.ac.uk

Marco Console
School of Informatics
University of Edinburgh
mconsole@inf.ed.ac.uk

Andreas Pieris
School of Informatics
University of Edinburgh
apieris@inf.ed.ac.uk

ABSTRACT

Consistent query answering (CQA) aims to deliver meaningful answers when queries are evaluated over inconsistent databases. Such answers must be certainly true in all repairs, which are consistent databases whose difference from the inconsistent one is somehow minimal. An interesting task in this context is to count the number of repairs that entail the query. This problem has been already studied for conjunctive queries and primary keys; we know that it is #P-complete in data complexity under polynomial-time Turing reductions (a.k.a. Cook reductions). However, as it has been already observed in the literature of counting complexity, there are problems that are "hard-to-count-easy-to-decide", which cannot be complete (under reasonable assumptions) for #P under weaker reductions, and, in particular, under standard many-one logspace reductions (a.k.a. parsimonious reductions). For such "hard-to-count-easy-to-decide" problems, a crucial question is whether we can determine their exact complexity by looking for subclasses of #P to which they belong. Ideally, we would like to show that such a problem is complete for a subclass of #P under many-one logspace reductions. The main goal of this work is to perform such a refined analysis for the problem of counting the number of repairs under primary keys that entail the query.

CCS CONCEPTS

• **Information systems → Inconsistent data**; • **Theory of computation → Complexity classes**; **Numeric approximation algorithms**;

PODS'19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06…$15.00
https://doi.org/10.1145/3294052.3319703

KEYWORDS

inconsistency, repairs, primary keys, first-order queries, counting, complexity, approximation schemes

ACM Reference Format:
Marco Calautti, Marco Console, and Andreas Pieris. 2019. Counting Database Repairs under Primary Keys Revisited. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3294052.3319703

1 INTRODUCTION

Inconsistent databases, that is, databases that do not conform to their specifications in the form of integrity constraints, are a real-life phenomenon that arises due to many reasons such as integration of conflicting sources. With the aim of obtaining conceptually meaningful answers to queries posed over inconsistent databases, Arenas, Bertossi, and Chomicki introduced in the late 1990s the notion of Consistent Query Answering (CQA) [2]. The key elements of the CQA approach can be summarized as follows: (1) the notion of *repair* of an inconsistent database D, i.e., a consistent database whose difference with D is minimal, and (2) the notion of query answering based on *certain answers*, i.e., answers that are entailed by every repair. Here is a simple example.

Example 1.1. Consider the schema $\mathbf{S}$ consisting of a single relation Employee(id, name, dept) that comes with the key constraint key(Employee) = {1}, which states that the first attribute, i.e., the id, is the key of the relation Employee. Consider now the database D over $\mathbf{S}$ consisting of the atoms:

$$\text{Employee}(1, \text{Bob}, \text{HR}) \qquad \text{Employee}(1, \text{Bob}, \text{IT})$$
$$\text{Employee}(2, \text{Alice}, \text{IT}) \qquad \text{Employee}(2, \text{Tim}, \text{IT}).$$

Observe that the above database is inconsistent w.r.t. the key constraint since we are uncertain about Bob's department, and the name of the employee with id 2. In this case, to devise a repair, we only need to keep one of the two atoms that are in a conflict. In this way, we obtain a $\subseteq$-maximal subset of D that is consistent. Now, consider the Boolean query Q

$$\exists x \exists y \exists z (\text{Employee}(1, x, y) \wedge \text{Employee}(2, z, y)),$$

which asks whether employees 1 and 2 work in the same department. This query is true only in two repairs, and thus, according to the certain answer semantics, is not entailed. ■

The constraint used in the above example is a primary key. Primary keys, which means that each relation of the schema has at most one key, form a central class of constraints that is heavily used in real-life applications. As we have seen above, in the presence of primary keys, a repair of a database D is a maximal consistent subset of D. The rest of this work deals only with primary keys.

1.1 Counting Database Repairs

Example 1.1, despite its simplicity, illustrates one of the limitations of the standard CQA approach. The notion of certain answers only says that a tuple is entailed by all repairs, or is not entailed by some repair. But, as discussed in [4], the former is too strict, while the latter is not very useful in a more practical context. Instead, we would like to know how often a tuple is a consistent answer, i.e., its relative frequency. For instance, in Example 1.1, the relative frequency of the query (in fact, of the empty tuple) is $\frac{1}{2}$ since, out of four repairs in total, two of those entail the query. Of course, to compute the relative frequency of a tuple, we need a way to compute (i) the number of repairs that entail a tuple (the numerator), and (ii) the total number of repairs (the denominator).

It is easy to show that, assuming a fixed query and set of primary keys (i.e., focussing on the data complexity), computing the total number of repairs is an easy task, i.e., feasible in polynomial time; see, e.g., [8]. However, computing the number of repairs that entail a given tuple is #P-complete in data complexity, under polynomial-time Turing reductions, when we focus on conjunctive queries [8, 9]. Recall that #P is a hard counting complexity class, which, roughly speaking, collects the function problems that ask for the number of solutions to an NP problem. Maslowski and Wijsen analyzed further this counting problem. They proved in [8] that, for every self-join-free conjunctive query and set of primary keys, it is either tractable, i.e., in FP, or #P-hard under polynomial-time Turing reductions, and they lifted this result to conjunctive queries with self-joins in [9].

1.2 Our Main Goal

From the above brief discussion, it is fair to conclude that, at least for conjunctive queries, the data complexity of the problem of counting the number of repairs that entail a given tuple is rather well-understood. Recall, however, that all the aforementioned completeness results rely on polynomial-time Turing reductions, a.k.a. Cook reductions. As it has been already observed in the literature (see, e.g., [6, 10]), there are problems that are "hard-to-count-easy-to-decide", which cannot be complete (under reasonable assumptions) for #P under standard many-one logspace (or even polynomial-time) reductions; these reductions are also called parsimonious. Note that the decision version of a counting problem asks whether the number of solutions is greater than zero. In our case, it asks whether there is a repair that entails a given tuple.

As stated in [10], Cook reductions blur structural differences between counting problems and complexity classes. The reason is that #P is not closed under Cook reductions (under reasonable assumptions). Therefore, for such "hard-to-count-easy-to-decide" problems, a crucial question is whether we can determine their exact complexity by looking for subclasses of #P to which they belong. Ideally, we would like to show that such a problem is complete for a subclass C of #P under many-one logspace reductions, where C is closed under many-one logspace reductions.

The main goal of this work is to perform such a refined complexity analysis for the problem of counting the number of repairs that entail a given tuple, always focussing on its data complexity. The query can be an arbitrary first-order query, or a query from important fragments such as positive existential queries, while the constraints are primary keys.

1.3 Research Challenges

After a preliminary complexity analysis, we have concluded that in the case of arbitrary first-order queries, our problem is not only "hard-to-count", but also "hard-to-decide". We can show that the decision problem is NP-complete, while the counting problem is #P-complete, both under many-one logspace reductions. This tells us that #P is the right complexity for our problem when we focus on first-order queries.

On the other hand, the situation for positive existential queries is more interesting. We can show that in this case we are dealing with a typical "hard-to-count-easy-to-decide" problem, and therefore, further investigation is needed. We know several counting complexity classes inside #P. One of those, which is central for our work, is SpanL, which, roughly speaking, collects all the functions that compute the number of distinct accepted outputs of a nondeterministic logspace transducer [1]. Although we can place our problem inside SpanL, it is unlikely to be complete for SpanL under many-one logspace reductions since this implies that L = NL. In view of the fact that we do not know much about subclasses of SpanL for which our problem can be complete under logspace reductions, this brings us to our first question:

Question 1: *Can we isolate a complexity class $C \subseteq$ SpanL, which is closed under many-one logspace reductions, for which our problem is complete under many-one logspace reductions?*

At this point, let us stress one more time that we focus on the data complexity of our problem. This means, by convention, that we deal, not with a single problem, but with a family of problems; as usual, each query and set of primary

keys gives rise to a new problem. Furthermore, establishing a completeness result for a certain complexity class C boils down to showing that (i) every problem in this family belongs to C, and (ii) there exists one problem that is C-complete. But there is no guarantee that there exists a problem in this family such that all the other problems of the family are logspace reducible to it. This means that there is no guarantee that the answer to the above question is affirmative. This brings us to our second research question, which concerns the case where Question 1 is answered negatively:

Question 2: *Can we characterize the data complexity of a parameterized version of our problem, i.e., when the query and the set of primary keys enjoy additional properties?*

Such a property can be, for example, a constant bound on the keywidth of the query w.r.t. the set of primary keys, i.e, the number of atoms occurring in the query with a relation that has a key. Indeed, as we shall see, the notion of keywidth plays a critical role in our analysis.

1.4 Summary of Contributions

Our main results can be summarized as follows; let us clarify that all the complexity results refer to the data complexity:

▶ In Section 3, we perform a preliminary complexity analysis under many-one logspace reductions. We show that our problem is "hard-to-count-hard-to-decide" for arbitrary first-order queries, but "hard-to-count-easy-to-decide" for positive existential queries. For the latter case, we further show that our problem is in SpanL, which is essentially the starting point of our subsequent investigation.

▶ In Section 4, we isolate a hierarchy of complexity classes inside SpanL, dubbed the Λ-hierarchy. The main technical challenge here is to understand how to limit the power of logspace nondeterministic transducers in such a way that will lead to the desired complexity classes. Note that the Λ-hierarchy, apart from the 0th and 1st levels, consists of hard complexity classes. In particular, if the second level is inside polynomial-time, then P = NP.

▶ We then proceed, in Section 5, to show that our problem, assuming that the keywidth of the query w.r.t. the set of primary keys is bounded by $k \geq 0$, is $\Lambda[k]$-complete under many-one logspace reductions, where $\Lambda[k]$ denotes the kth level of the Λ-hierarchy. This provides a firm answer to our second question above.

▶ By exploiting the above result, we can partially answer our first question. In Section 5, we show that: either our problem is $\Lambda[k]$-complete, for some $k \geq 0$, or there is no class $C \subseteq$ SpanL, which is closed under logspace reductions, for which our problem is complete; both statements are under many-one logspace reductions. As the reader may have already suspected, the existence of an integer $k \geq 0$ such that $\Lambda[k]$ is the right complexity class for our problem is equivalent to the fact that the Λ-hierarchy collapses. The latter is a difficult question that remains open.

▶ Since our problem is hard, it is natural to try to approximate it. In Section 6, we target fully polynomial-time randomized approximation schemes, FPRAS for short. Although for arbitrary first-order queries our problem does not admit an FPRAS (under reasonable assumptions), in the case of positive existential queries an FPRAS can be devised. Actually, this is not a new result since we can inherit the FPRAS for query answering over disjoint-independent probabilistic databases [5]. The novelty lies in the fact that our FPRAS is conceptually simpler than the one from [5]. Note that this result is obtained by showing a more general result, i.e., every function in $\Lambda[k]$, for $k > 1$, admits an FPRAS.

▶ Finally, in Section 7, we provide more completeness results for the Λ-hierarchy, which justifies even further its meaningfulness. In particular, we show that counting the satisfying assignments of disjoint positive kDNF formulas, as well as counting the number of forbidden colorings of k-uniform hypergraphs, are $\Lambda[k]$-complete.

2 PRELIMINARIES

In this section, we recall the basics on relational databases, key constraints, and queries. We also recall some fundamental notions from complexity theory.

2.1 Databases, Constraints and Queries

Relational Databases. We assume a countably infinite set $\mathbf{C}$ of *constants* from which database elements are drawn. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity n. A *fact* over $\mathbf{S}$ is an expression of the form $R(c_1, \ldots, c_n)$, where $R/n \in \mathbf{S}$ with $n > 0$, and $c_i \in \mathbf{C}$ for each $i \in \{1, \ldots, n\}$. A *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of a database D, denoted $\mathrm{dom}(D)$, is the set of constants occurring in D.

Primary Key Constraints. A *key constraint* (or simply *key*) κ over a schema $\mathbf{S}$ is an expression of the form $\mathrm{key}(R) = A$, where $R/n \in \mathbf{S}$ and $A \subseteq \{1, \ldots, n\}$. Such an expression is called an R-key. A database D satisfies κ if, for every two facts $R(\bar{t}), R(\bar{s}) \in D$, $\bar{t}[A] = \bar{s}[A]$ implies $\bar{t} = \bar{s}$. We say that D is *consistent* w.r.t. a set Σ of keys, written $D \models \Sigma$, if D satisfies each key in Σ; otherwise, is *inconsistent* w.r.t. Σ. In this work, we focus on sets of *primary keys*, i.e., sets of keys that, for each predicate R of the underlying schema, have at most one R-key. For technical clarity, we assume, w.l.o.g., that each key constraint is of the form $\mathrm{key}(R) = \{1, \ldots, m\}$, i.e., always the first m attributes of R/n, for some $m \leq n$, form the key.

Queries. Let $\mathbf{V}$ be a countably infinite set of *variables* disjoint from $\mathbf{C}$. An atom over a schema $\mathbf{S}$ is an expression of

the form $R(t_1, \ldots, t_n)$, where $R/n \in \mathbf{S}$, and $t_i \in \mathbf{C} \cup \mathbf{V}$ for each $i \in \{1, \ldots, n\}$. Notice that a fact is an atom without variables. A *first-order query* $Q(\bar{x})$ over a schema $\mathbf{S}$ is an expression $\{\bar{x} \mid \varphi\}$, where φ is a first-order formula with free variables $\bar{x}$ that mentions only atoms over $\mathbf{S}$. The class of first-order queries is denoted $\mathbb{FO}$. We are also going to consider key fragments of $\mathbb{FO}$, in particular, *existential positive queries* ($\exists\mathbb{FO}^+$), *conjunctive queries* ($\mathbb{CQ}$), and *unions of conjunctive queries* ($\mathbb{UCQ}$). The answer to $Q(\bar{x})$ over a database D, denoted $Q(D)$, is the set of tuples $\{\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|} \mid D \models \varphi(\bar{c})\}$.

Database Repairs and Blocks. For an inconsistent database D w.r.t. a set Σ of primary keys, a *repair* of D w.r.t. Σ is a maximal subset of D that is consistent w.r.t. Σ; i.e., if $D' \subseteq D$ is a repair, then there is no $D'' \subseteq D$ that is consistent w.r.t. Σ and $D' \subsetneq D''$. Let $\mathsf{rep}(D, \Sigma)$ be the set of repairs of D w.r.t Σ.

When we focus on primary keys, there is a convenient way to construct repairs. We first collect all the facts of the database that are in a conflict, i.e., they have the same predicate R and agree on $\mathsf{key}(R)$, into disjoint sets called blocks. A repair can then be constructed by keeping exactly one fact from each block. Formally, for a fact $\alpha = R(c_1, \ldots, c_n)$, the *key value* of α w.r.t. Σ is defined as

$$\mathsf{key}_\Sigma(\alpha) = \begin{cases} \langle R, \langle c_1, \ldots, c_m \rangle \rangle & \text{if } \mathsf{key}(R) = \{1, \ldots, m\} \in \Sigma, \\ \langle R, \langle c_1, \ldots, c_n \rangle \rangle & \text{otherwise.} \end{cases}$$

Then, given a database D, we define

$$\mathsf{block}_\Sigma(\alpha, D) = \{\beta \in D \mid \mathsf{key}_\Sigma(\beta) = \mathsf{key}_\Sigma(\alpha)\}.$$

For $\alpha, \beta \in D$, if $\mathsf{key}_\Sigma(\alpha) = \mathsf{key}_\Sigma(\beta)$, then $\mathsf{block}_\Sigma(\alpha, D)$ and $\mathsf{block}_\Sigma(\beta, D)$ coincide. Moreover, $\mathsf{block}_\Sigma(\alpha, D) = \{\alpha\}$ means that α is not in a conflict. A repair can be constructed by keeping one fact from each block. Formally, with $\mathsf{block}_\Sigma(D) = \{\mathsf{block}_\Sigma(\alpha, D) \mid \alpha \in D\}$ and $\Pi_{D,\Sigma} = \bigtimes_{B \in \mathsf{block}_\Sigma(D)} B$, it holds that $\mathsf{rep}(D, \Sigma) = \{\{\alpha_1, \ldots, \alpha_n\} \mid \langle \alpha_1, \ldots, \alpha_n \rangle \in \Pi_{D,\Sigma}\}$.

For a database D and set Σ of primary keys, we write $\prec_{D,\Sigma}$ for the lexicographic ordering over the set of key values $\{\mathsf{key}_\Sigma(\alpha) \mid \alpha \in D\}$. Observe that $\prec_{D,\Sigma}$ induces a sequence $B_1, \ldots, B_n$ of the sets of $\mathsf{block}_\Sigma(D)$: assuming that $\kappa_1 \prec_{D,\Sigma} \cdots \prec_{D,\Sigma} \kappa_n$, the key value of the facts of B_i is κ_i. Note that in case D is empty, the above sequence is empty, i.e., $n = 0$.

Counting Database Repairs. An interesting problem, which has been already studied in [8, 9], and is the main concern of this work, is the following; $\mathbb{Q}$ is a class of queries:

PROBLEM :	$\#\mathsf{CQA}(\mathbb{Q})$
INPUT :	A database D, a set Σ of primary keys, $Q(\bar{x}) \in \mathbb{Q}$, and a tuple $\bar{t} \in \mathsf{dom}(D)^{\lvert\bar{x}\rvert}$.
OUTPUT :	$\lvert\{D' \in \mathsf{rep}(D, \Sigma) \mid \bar{t} \in Q(D')\}\rvert$.

The goal of this work is to revisit the above problem with the aim of providing more insights about its computational complexity. As usual, we are interested in its *data complexity*, where the set of constraints and the query are fixed, and only the database and the tuple are considered input. Let us make this more precise as it is very crucial for our work. For every pair of a query $Q(\bar{x}) \in \mathbb{Q}$ and a set Σ of primary keys, we can define the following problem:

PROBLEM :	$\#\mathsf{CQA}(Q(\bar{x}), \Sigma)$
INPUT :	A database D, and a tuple $\bar{t} \in \mathsf{dom}(D)^{\lvert\bar{x}\rvert}$.
OUTPUT :	$\lvert\{D' \in \mathsf{rep}(D, \Sigma) \mid \bar{t} \in Q(D')\}\rvert$.

Let $\mathcal{C}$ be a complexity class, and $\mathcal{R}$ a class of reductions; details about complexity classes and reductions are given below. We say that $\#\mathsf{CQA}(\mathbb{Q})$ is $\mathcal{R}$-complete for $\mathcal{C}$ in data complexity if: (i) for each $Q \in \mathbb{Q}$ and set Σ of primary keys, $\#\mathsf{CQA}(Q, \Sigma)$ is in $\mathcal{C}$, and (ii) there is $Q' \in \mathbb{Q}$ and a set Σ' of primary keys such that $\#\mathsf{CQA}(Q', \Sigma')$ is $\mathcal{R}$-complete for $\mathcal{C}$.

The above counting problems have a natural decision version, denoted $\#\mathsf{CQA}_{>0}(\mathbb{Q})$ and $\#\mathsf{CQA}_{>0}(Q, \Sigma)$, respectively, that simply asks whether $|\{D' \in \mathsf{rep}(D, \Sigma) \mid \bar{t} \in Q(D')\}| > 0$. The data complexity of $\#\mathsf{CQA}_{>0}(\mathbb{Q})$ is defined in the same way as for $\#\mathsf{CQA}(\mathbb{Q})$. Henceforth, we focus on Boolean queries, but all the results extend to non-Boolean queries.

2.2 Complexity Toolbox

Counting Complexity Classes. We recall well-known counting complexity classes, i.e., classes of counting functions of the form $\{0,1\}^* \to \mathbb{N}$, that are crucial for our analysis. We first recall the counting analog of polynomial-time:

$$\mathrm{FP} = \{f \mid f \text{ is computable in polynomial-time}\}.$$

Given a nondeterministic Turing machine (NTM) M, let accept_M be the function $\{0,1\}^* \to \mathbb{N}$ such that $\mathsf{accept}_M(x)$ is the number of accepting computations of M on input x. We can then define:

$$\#\mathrm{P} = \{\mathsf{accept}_M \mid M \text{ is a polynomial-time NTM}\}$$
$$\#\mathrm{L} = \{\mathsf{accept}_M \mid M \text{ is a logarithmic-space NTM}\}.$$

Given an NTM M with output tape, i.e., a nondeterministic transducer (NTT), let span_M be the function $\{0,1\}^* \to \mathbb{N}$ such that $\mathsf{span}_M(x)$ is the number of *distinct valid* outputs of M on input x; the output of a computation is considered to be valid if the machine halts in an accepting state. Then:

$$\mathrm{SpanL} = \{\mathsf{span}_M \mid M \text{ is a logarithmic-space NTT}\}.$$

For the above classes we know that $\#\mathrm{L} \subseteq \mathrm{FP}, \mathrm{SpanL} \subseteq \#\mathrm{P}$. It is also believed that the above inclusions are strict (under reasonable complexity-theoretic assumptions) [1].

Let us recall that every counting problem f has a natural decision version, denoted by $f_{>0}$, defined as expected: given $x \in \{0,1\}^*$, is it the case that $f(x) > 0$? Actually, we have

already seen this for #CQA($\mathbb{Q}$) and #CQA(Q, Σ) in the previous section. Of course, for pinpointing the complexity of $f_{>0}$ we rely on standard complexity classes.

Reductions. Consider two functions $f, g : \{0,1\}^* \to \mathbb{N}$. We say that f is *Cook reducible* to g, denoted $f \leq_{\mathrm{T}}^{\mathrm{p}} g$, if there exists a polynomial-time deterministic transducer M, with access to an oracle for g, such that, for every $x \in \{0,1\}^*$, $f(x) = M(x)$. In other words, a (functional) Cook reduction is a polynomial-time (functional) Turing reduction. Moreover, we say that f is *many-one logspace reducible* to g, denoted $f \leq_{\mathrm{m}}^{\log} g$, if there is a function $h : \{0,1\}^* \to \{0,1\}^*$ that is computable in logarithmic-space such that, for every $x \in \{0,1\}^*$, $f(x) = g(h(x))$. Let us say that when referring to counting functions, many-one reductions are also called *parsimonious* reductions. A class C is *closed* under Cook (resp., many-one logspace) reductions if, for every two functions f, g, $f \leq_{\mathrm{T}}^{\mathrm{p}} g$ (resp., $f \leq_{\mathrm{m}}^{\log} g$) and $g \in C$ implies $f \in C$.

3 COMPLEXITY OF #CQA: A GLIMPSE

The complexity of #CQA has been already studied by Maslowski and Wijsen in [8, 9]. Notice that all the complexity results for #CQA and #CQA$_{>0}$ that will be given in the rest of the paper concern their data complexity. Thus, for brevity, we will sometimes avoid to explicitly mention the term data complexity. We know that:

Theorem 3.1 ([8, 9]). #CQA($\mathbb{CQ}$) *is* $\leq_{\mathrm{T}}^{\mathrm{p}}$*-complete for* #P.

The above result relies on Cook reductions. However, as it has been already discussed in Section 1, there are problems that are "hard-to-count-easy-to-decide", which cannot be complete (under reasonable assumptions) for #P under many-one logspace (or even polynomial-time) reductions. For such problems, a crucial question is whether we can determine their exact complexity by looking for subclasses of #P to which they belong. Our main goal here is to perform such an analysis for #CQA($\mathbb{Q}$), where $\mathbb{Q}$ is one of the standard query languages mentioned in Section 2.

3.1 Arbitrary First-Order Queries

Is #CQA($\mathbb{FO}$) an "easy-to-decide" problem? Unfortunately, the answer to this question is negative. We can show that:

Theorem 3.2. #CQA$_{>0}$($\mathbb{FO}$) *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.

For the upper bound (assume a fixed query $Q \in \mathbb{FO}$ and a set Σ of primary keys), given a database D, we simply need to guess a database $D' \subseteq D$, and then verify that $D' \in \mathsf{rep}(D, \Sigma)$ and $D' \models Q$. It is clear that this nondeterministic procedure runs in polynomial time. For the lower bound, we show that there is a query $Q \in \mathbb{FO}$ and a set Σ of primary keys such that 3SAT $\leq_{\mathrm{m}}^{\log}$ #CQA$_{>0}(Q, \Sigma)$.

With Theorem 3.2 in place, it is not surprising that #CQA($\mathbb{FO}$) is also "hard-to-count". It is easy to show that:

Theorem 3.3. #CQA($\mathbb{FO}$) *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* #P.

Proof. For the lower bound, since #3SAT, i.e., the counting version of 3SAT, is $\leq_{\mathrm{m}}^{\log}$-complete for #P [11], we can exploit the construction given in the proof of Theorem 3.2. For the upper bound, fix a query $Q \in \mathbb{FO}$ and a set Σ of primary keys. We need to show that #CQA(Q, Σ) is a function of the form accept$_M$, where M is a polynomial-time NTM. Consider a database D, and let $B_1, \ldots, B_n$ be the sequence of blocks induced by $\prec_{D,\Sigma}$. The machine M, on input D, performs two simple steps: (1) for $i = 1$ to n: guess a fact $\alpha_i \in B_i$, and (2) if $\{\alpha_1, \ldots, \alpha_n\} \models Q$, then accept; otherwise, reject. It is clear that $D' = \{\alpha_1, \ldots, \alpha_n\} \in \mathsf{rep}(D, \Sigma)$. Moreover, since D' is constructed according to a fixed ordering, each computation of M on input D builds a distinct repair of $\mathsf{rep}(D, \Sigma)$. Thus, the number of accepting computations of M on D is the number of repairs of $\mathsf{rep}(D, \Sigma)$ that entail Q. Since the machine M runs in polynomial time, the claim follows. □

Since #P is closed under many-one logspace reductions, Theorem 3.3 implies that #P is the right complexity for #CQA($\mathbb{FO}$), and thus, there is no room for further analysis.

3.2 Existential Positive First-Order Queries

The situation for existential positive queries is more interesting. According to the following result, we are dealing with a typical "hard-to-count-easy-to-decide" problem.

Theorem 3.4. #CQA$_{>0}$($\exists\mathbb{FO}^+$) *is in* L.

Proof. Fix a query $Q \in \exists\mathbb{FO}^+$ and a set Σ of primary keys. We need to show that #CQA$_{>0}(Q, \Sigma)$ is in L. We first recall that Q can be equivalently rewritten in constant time (since the query is not part of the input) as a query $Q' \in \mathbb{UCQ}$ of the form $\bigvee_{i=1}^{n} Q_i$, where each Q_i is a Boolean conjunctive query. Thus, it suffices to show that #CQA$(Q', \Sigma)_{>0}$ is in L. The crucial observation, which is easy to verify, is the following; we write $\mathsf{var}(Q_i)$ for the variables in Q_i:

Lemma 3.5. *Given a database* D*, there exists* $D' \in \mathsf{rep}(D, \Sigma)$ *such that* $D' \models Q'$ *iff there exists* $i \in \{1, \ldots, n\}$ *and a mapping* h *from* $\mathsf{var}(Q_i)$ *to* $\mathsf{dom}(D)$ *such that* $h(Q_i) \subseteq D$ *and* $h(Q_i) \models \Sigma$.

We then get the following logspace procedure: if there is $i \in \{1, \ldots, n\}$ and mapping $h : \mathsf{var}(Q_i) \to \mathsf{dom}(D)$ such that $h(Q_i) \subseteq D$ and $h(Q_i) \models \Sigma$, then accept; otherwise, reject. □

The following is an easy consequence of Theorem 3.4:

Proposition 3.6. *If* #CQA($\exists\mathbb{FO}^+$) *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* #P*, then* P = NP. *This holds even for polynomial-time reductions.*

Input: a database D
Output: a member of $\text{rep}(D, \Sigma)$

guess $i \in \{1, \ldots, m\}$ and mapping
$h : \text{var}(Q_i) \to \text{dom}(D)$;
if $h(Q_i) \nsubseteq D$ or $h(Q_i) \not\models \Sigma$ **then**
  reject;
$j := 1$;
while $j \leq n$ **do**
  if $B_j \cap h(Q_i) = \{R(\bar{t})\}$ and Σ has an R-key **then**
    output $R(\bar{t})$;
  else
    guess $\alpha \in B_j$;
    output α;
  $j := j + 1$;
accept.

Algorithm 1: The nondeterministic transducer $M_{Q,\Sigma}$.

Towards pinpointing the complexity of $\#\text{CQA}(\exists\mathbb{FO}^+)$, Proposition 3.6 tells us that first we should look for a subclass of #P, closed under many-one logspace reductions, in which we can place our problem. The obvious candidate is SpanL:

Theorem 3.7. $\#\text{CQA}(\exists\mathbb{FO}^+)$ *is in* SpanL.

Proof. It suffices to show that $\#\text{CQA}(\mathbb{UCQ})$ is in SpanL. Fix a query $Q \in \mathbb{UCQ}$ of the form $\bigvee_{i=1}^{m} Q_i$, and a set Σ of primary keys. We need to show that $\#\text{CQA}(Q, \Sigma)$ is a function of the form span_M, where M is a logarithmic-space NTT. Consider the transducer $M_{Q,\Sigma}$, depicted in Algorithm 1, that accepts as input a database D, and outputs a member of $\text{rep}(D, \Sigma)$. Let $B_1, \ldots, B_n$ be the sequence of blocks from $\text{block}_\Sigma(D)$ induced by $\prec_{D,\Sigma}$. It can be shown that $M_{Q,\Sigma}$ uses only logarithmic-space. It is also not difficult to see that (1) for each $D' \in \text{rep}(D, \Sigma)$, $D' \models Q$ iff there exists an accepting computation of $M_{Q,\Sigma}$ on D that outputs D', and (2) for each output $\alpha_1, \ldots, \alpha_n$ of an accepting computation of $M_{Q,\Sigma}$ on D, $\alpha_i \in B_i$, i.e., facts from a certain block always appear at the same position of the output. The latter statement ensures that it is not possible for two different accepting computations to produce the same repair, while their output strings are syntactically different by producing the same facts in different order. Hence, from (1) and (2), we get that the number of distinct valid outputs of $M_{Q,\Sigma}$ on input D is exactly the number of repairs of $\text{rep}(D, \Sigma)$ that entail Q. □

At this point, one may be tempted to think that SpanL is the right complexity for $\#\text{CQA}(\exists\mathbb{FO}^+)$. Unfortunately, the situation is more complex. The following consequence of Theorem 3.4 states that it is unlikely to be the case:

Proposition 3.8. *If* $\#\text{CQA}(\exists\mathbb{FO}^+)$ *is* $\leq_{\text{m}}^{\log}$*-complete for* SpanL*, then* L = NL.

What about subclasses of SpanL? From Section 2.2, we know that #L ⊆ SpanL. However, since this class is tractable, while our problem is hard (i.e., $\leq_{\text{T}}^{\text{p}}$-complete for #P), it is again unlikely that this is the class that we are looking for.

Proposition 3.9. *If* $\#\text{CQA}(\exists\mathbb{FO}^+)$ *is in* #L*, then* P = NP.

Note that the above holds even if we replace #L with FP.

Our Main Research Questions

The above discussion brings us to our main questions, already discussed in Section 1. We proceed to expand a bit more on those questions in order to make them more precise.

Question 1: *Is there a class* $C \subseteq$ SpanL*, closed under logspace reductions, such that* $\#\text{CQA}(\exists\mathbb{FO}^+)$ *is* $\leq_{\text{m}}^{\log}$*-complete for* C*?*

One may think that the above question is trivial since, for every function $f \in$ SpanL, there is a class $C \subseteq$ SpanL, closed under logspace reductions, such that f is $\leq_{\text{m}}^{\log}$-complete for C: define C as $\{g \in \text{SpanL} \mid g \leq_{\text{m}}^{\log} f\}$. However, it should not be forgotten that we concentrate on the data complexity of $\#\text{CQA}(\exists\mathbb{FO}^+)$, which means that we deal with a family F of functions and not a single function, and, by convention, F is $\leq_{\text{m}}^{\log}$-complete for C if (i) $\mathsf{F} \subseteq C$, and (ii) there is a function of F that is $\leq_{\text{m}}^{\log}$-complete for C. But there is no guarantee that there is $f \in \mathsf{F}$ such that, for every $g \in \mathsf{F}$, $g \leq_{\text{m}}^{\log} f$. This means that it might be the case that a class $C \subseteq$ SpanL, closed under logspace reductions, for which $\#\text{CQA}(\exists\mathbb{FO}^+)$ is $\leq_{\text{m}}^{\log}$-complete does not exist. This brings us to our second question, which is a refined version of the first one:

Question 2: *Can we characterize the data complexity of a parameterized version of* $\#\text{CQA}(\exists\mathbb{FO}^+)$*?*

Let us make this more precise. A *covering function for* $\#\text{CQA}(\exists\mathbb{FO}^+)$ is a function that maps pairs of the form (Q, Σ), where Q is a query from $\exists\mathbb{FO}^+$ and Σ is a set of primary keys, to the natural numbers. Such a function *cov* is called covering for $\#\text{CQA}(\exists\mathbb{FO}^+)$ since it induces a cover for $\mathsf{F} = \{\#\text{CQA}(Q, \Sigma) \mid Q \in \exists\mathbb{FO}^+ \text{ and } \Sigma \text{ is a set of primary keys}\}$, i.e., the family of problems that we have to deal with in order to pinpoint the data complexity of $\#\text{CQA}(\exists\mathbb{FO}^+)$. More precisely, *cov* induces a sequence $\mathsf{F}_0, \mathsf{F}_1, \ldots$ of subsets of F, where $\mathsf{F}_k = \{\#\text{CQA}(Q, \Sigma) \mid \textit{cov}(Q, \Sigma) = k\}$, that covers F in the sense that $\mathsf{F} = \bigcup_{i \geq 0} \mathsf{F}_i$. We can now consider a parameterized version of $\#\text{CQA}(\exists\mathbb{FO}^+)$ defined as follows:

PROBLEM :	$\#\text{CQA}_k^{cov}(\exists\mathbb{FO}^+)$
INPUT :	A database D, a set Σ of primary keys, and $Q \in \exists\mathbb{FO}^+$ such that $\textit{cov}(Q, \Sigma) = k$.
OUTPUT :	$\lvert\{D' \in \text{rep}(D, \Sigma) \mid D' \models Q\}\rvert$.

The data complexity of the above problem is defined as expected. In case Question 1 is answered negatively, the best

that we can hope for is to isolate a covering function *cov* that allows us to pinpoint the data complexity of $\#\mathsf{CQA}_k^{cov}(\exists\mathbb{F}\mathbb{O}^+)$. Question 2 asks whether such a covering function exists.

Plan of Attack. We will first focus on our second question and provide a definite answer, which will then allow us to partially answer the first question. Despite our efforts, we have not managed to provide a firm answer to the first question. As we shall see, it is equivalent to a problem concerning the collapsing of a hierarchy of complexity classes inside SpanL. Our plan is as follows:

▶ We show that the *keywidth function* (*kw*), which computes the number of atoms in Q that use a predicate with a key (w.r.t. to Σ), is a covering function that allows us to pinpoint the complexity of $\#\mathsf{CQA}_k^{kw}(\exists\mathbb{F}\mathbb{O}^+)$. To this end, we first isolate (in Section 4) a hierarchy of classes inside SpanL, which are closed under logspace reductions, dubbed the Λ-*hierarchy*. We then show (in Section 5) that, for each $k \geq 0$, $\#\mathsf{CQA}_k^{kw}(\exists\mathbb{F}\mathbb{O}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\Lambda[k]$, i.e., the kth level of the Λ-hierarchy.

▶ Then, we proceed (in Section 5) to partially answer our first question. By exploiting the above result, we show the following: either $\#\mathsf{CQA}(\exists\mathbb{F}\mathbb{O}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\Lambda[k]$, for some $k \geq 0$, or there is no $C \subseteq$ SpanL, closed under logspace reductions, such that $\#\mathsf{CQA}(\exists\mathbb{F}\mathbb{O}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for C. The existence of an integer $k \geq 0$ such that $\Lambda[k]$ is the right complexity class for $\#\mathsf{CQA}(\exists\mathbb{F}\mathbb{O}^+)$ is equivalent to the fact that the Λ-hierarchy collapses. Whether the latter holds seems to be a difficult question that remains open.

4 THE Λ-HIERARCHY

We now proceed to introduce our hierarchy of complexity classes. The main technical challenge is to understand how to limit the computational power of logarithmic-space NTTs, which in turn will lead to our new complexity classes inside SpanL that serve the purpose described in our plan of attack. To this end, we first give a semi-formal discussion on a couple of structural properties that our problem enjoys, which will provide important insights on how to overcome our main technical challenge. Interestingly, these properties are shared by several natural problems inside SpanL.

4.1 The Guess-Check-Expand Paradigm

It turned out that there exists a generic paradigm, which we call *guess-check-expand*, that a logspace NTT can follow in order to place problems inside SpanL, assuming that the following structural properties are fulfilled:

Small Certificates. A solution (in our case, a repair that entails the query) is witnessed via a *small certificate*, i.e., of logarithmic size, that is verifiable in logspace.

For the next property, we first need to give some auxiliary terminology. Given a sequence of sets $S_1, \ldots, S_n$, an ℓ-*selector* of it, for $\ell \geq 0$, is a sequence of pairs $\sigma = (i_1, e_1), \ldots, (i_\ell, e_\ell)$, where $1 \leq i_1 < \cdots < i_\ell \leq n$, and $e_j \in S_{i_j}$ for each $j \in \{1, \ldots, \ell\}$. Intuitively, a pair (i, e) in σ tells us to "keep the element e from the set S_i". We can then define the *cartesian product of* $S_1, \ldots, S_n$ *w.r.t.* σ, denoted $[S_1, \ldots, S_n]^\sigma$, as $S_1^\sigma \times \cdots \times S_n^\sigma$, where $S_i^\sigma = \{e\}$ if σ contains a pair (i, e), otherwise, $S_i^\sigma = S_i$. In simple words, $[S_1, \ldots, S_n]^\sigma$ is the cartesian product of $S_1, \ldots, S_n$ with the difference that ℓ sets are first replaced by a singleton set as dictated by the ℓ-selector σ.

We can now discuss the second property. For brevity, for an input instance x, let $\mathrm{sol}(x)$ be the set of solutions, and $\mathrm{cert}(x)$ the set of certificates that witness a solution of $\mathrm{sol}(x)$.

Solutions via Certificate Expansion. There is $k \geq 0$ such that, for every input instance x, there is a sequence of logspace computable sets $S_1, \ldots, S_n$, called *solution domains*, such that

$$|\mathrm{sol}(x)| = \left| \bigcup_{c \in \mathrm{cert}(x)} [S_1, \ldots, S_n]^{\sigma_c} \right|,$$

where σ_c is an ℓ-selector, with $\ell \leq k$, for $S_1, \ldots, S_n$ determined by the small certificate c. In fact, there is a bijection $enc : \mathrm{sol}(x) \to \bigcup_{c \in \mathrm{cert}(x)} [S_1, \ldots, S_n]^{\sigma_c}$, and $enc(s)$ should be seen as an encoding of the solution s.

A function that enjoys the above properties can be placed in SpanL via a simple guess-check-expand algorithm:

(1) (**Guess**) Guess a candidate small certificate c.
(2) (**Check**) If c is not a valid certificate, then reject.
(3) (**Expand**) Expand c into an encoding of a solution witnessed by c using the solution domains $S_1, \ldots, S_n$ as follows: for $i = 1$ to n, if the ℓ-selector σ_c determined by c mentions (i, e), then output e; otherwise, guess $e \in S_i$ and output e.

It is clear that, on an input x, the above NTT uses logarithmic space in the size of x since the small certificates are logspace verifiable, and the solution domains are logspace computable. It is also clear that the number of distinct valid outputs of the NTT coincides with $\left|\bigcup_{c \in \mathrm{cert}(x)} [S_1, \ldots, S_n]^{\sigma_c}\right|$, and thus, with $|\mathrm{sol}(x)|$, as needed.

$\#\mathsf{CQA}(Q, \Sigma)$ via a Guess-Check-Expand Algorithm. It is interesting to observe that the problem $\#\mathsf{CQA}(Q, \Sigma)$, for some UCQ Q and set Σ of primary keys, enjoys the two structural properties discussed above; recall that an existential positive query can be rewritten as a UCQ. In fact, on input D:

- A small certificate for a repair of $\mathrm{rep}(D, \Sigma)$ that entails Q, is a pair (Q', h), where Q' is a disjunct of Q, and $h : \mathrm{var}(Q') \to \mathrm{dom}(D)$ with $h(Q') \subseteq D$ and $h(Q') \models \Sigma$.

- The sequence of solution domains corresponds to the sequence $B_1, \ldots, B_n$ of the sets of $\text{block}_\Sigma(D)$ induced by $\prec_{D,\Sigma}$. The ℓ-selector $\sigma_{(Q',h)}$ for $B_1, \ldots, B_n$ determined by a certificate (Q', h) mentions the pair $(i, R(\bar{t}))$, i.e., it keeps the fact $R(\bar{t})$ from B_i, iff $h(Q') \cap B_i = \{R(\bar{t})\}$ and Σ has an R-key. The latter implies that ℓ is bounded by the maximum number of atoms with a predicate that has a key over all disjuncts of Q, which does not depend on the input database D. Clearly, the number of repairs of $\text{rep}(D, \Sigma)$ that entail Q is the number

$$\left| \bigcup_{(Q',h) \in \text{cert}(D)} [B_1, \ldots, B_n]^{\sigma_{(Q',h)}} \right|.$$

Therefore, the fact that $\#\text{CQA}(Q, \Sigma)$ is in SpanL can be shown via a guess-check-expand algorithm. Actually, the logspace NTT presented in Algorithm 1 is such an algorithm:

(1) (**Guess**) It first guesses a candidate certificate (Q', h).
(2) (**Check**) If $h(Q') \not\subseteq D$ or $h(Q') \not\models \Sigma$, then it rejects.
(3) (**Expand**) It expands (Q', h) into an encoding of a solution witnessed by (Q', h) as follows: for $i = 1$ to n, if $h(Q') \cap B_i = \{R(\bar{t})\}$ and Σ has an R-key, then outputs $R(\bar{t})$; otherwise, it guesses a fact $\alpha \in B_i$ and outputs α.

Ubiquity of the Guess-Check-Expand Paradigm. It turned out that there are several natural problems that enjoy the above structural properties, and thus, can be placed in SpanL via the guess-check-expand paradigm. Here we give a partial list of such problems, which ask for the number of:

- Satisfying assignments of a positive kDNF formula.
- Non-independent sets of an undirected graph.
- Non-3-colorings of an undirected graph.
- Non-vertex-covers of an undirected graph.

Let us stress that the above list is by no means exhaustive, which is a strong indication that the guess-check-expand paradigm is a fundamental algorithm design paradigm that deserves our attention.

Problems Beyond Guess-Check-Expand. At this point, one may think that the structural properties discussed above are trivial, i.e., they are fulfilled by every problem inside SpanL. In such a case, all the functions inside SpanL admit a guess-check-expand algorithm, and thus, it would be of little use towards understanding how to limit the computational power of logarithmic-space NTTs, which in turn will lead to our new complexity classes. The small certificate property implies that the decision version of a counting problem is in L. Actually, this is how Theorem 3.4, which states that $\#\text{CQA}_{>0}(\exists\mathbb{FO}^+)$ is in L, is shown. Thus, if a $\leq_{\text{m}}^{\log}$-complete problem for SpanL enjoys the small certificate property, then L = NL. Hence, it is unlikely that $\leq_{\text{m}}^{\log}$-complete problems for SpanL can be solved via guess-check-expand algorithms (this is shown in Theorem 4.3).

4.2 Guess-Check-Expand via Compactors

The previous discussion suggests that our new complexity classes inside SpanL, which will form the Λ-hierarchy, should be defined by relying on logarithmic-space NTTs whose computational power does not go beyond guess-check-expand algorithms. In other words, we should limit the nondeterminism of logspace NTTs in such a way that they can only implement algorithms that follow the guess-check-expand paradigm. However, it is not clear at all how this can be achieved, while keeping the definition of the obtained class of NTTs compact and elegant.

Towards such an elegant definition, we observe that the "solutions via certificate expansion" property essentially tells us the following. On an input x, assuming that $S_1, \ldots, S_n$ are the solution domains, for a valid certificate $c \in \text{cert}(x)$, we can deterministically compute in logspace a compact representation of $[S_1, \ldots, S_n]^{\sigma_c}$. This is simply the sequence of sets $S_1^{\sigma_c}, \ldots, S_n^{\sigma_c}$, while its unfolding

$$\text{unfolding}\left(S_1^{\sigma_c}, \ldots, S_n^{\sigma_c}\right) \;=\; S_1^{\sigma_c} \times \cdots \times S_n^{\sigma_c}$$

gives us back the set $[S_1, \ldots, S_n]^{\sigma_c}$. Having this simple observation in place, we can implement guess-check-expand algorithms via deterministic logspace transducers, called *compactors*, which are responsible for computing compact representations as the one above. Somehow, a logspace compactor implements the deterministic part of a guess-check-expand algorithm, while the nondeterministic part is taken care by the unfolding of the compact representations. As we shall see, logspace compactors, together with the notion of unfolding, will allow us to provide an elegant definition for our hierarchy. Before giving the formal definitions, let us give some more details how compactors work.

Logspace Compactors and Unfolding. Consider a function f that enjoys the two main properties. Let x be an input instance, and $S_1, \ldots, S_n$ the solution domains. A logspace compactor M_f accepts as input x and a candidate certificate c, and performs two steps:

(1) (**Check**) If c is not valid, then output ϵ and halt.
(2) (**Compact**) For $i = 1$ to n, if the selector σ_c determined by c mentions (i, e), then output $\{e\}$; else, output S_i.

Let $\text{unfolding}(\epsilon) = \emptyset$, and assume that $\text{ccert}(x)$ collects all the candidate small certificates. It is not difficult to see that

$$f(x) \;=\; \left| \bigcup_{c \in \text{ccert}(x)} \text{unfolding}\left(M_f(x, c)\right) \right|.$$

Note that $\text{unfolding}(\epsilon) = \emptyset$ reflects the fact that invalid candidate certificates should not be taken into count.

4.3 The Λ-hierarchy: Definition and Results

The previous, rather extensive, semi-formal discussion provides important insights on how to limit the power of logspace NTTs, which in turn will lead to our new complexity classes that form the Λ-hierarchy. This will be done via logspace compactors and unfolding as sketched above. We proceed with the formal definitions.

Shape of Compact Representations. We first need to fix the syntactic shape of compact representations, and formalize the notion of unfolding. Let $S_1, \ldots, S_n \subseteq \{0,1\}^*$ be non-empty sets of strings, where $n \geq 0$; assume that, for $i \in \{1, \ldots, n\}$, $S_i = \{s_i^1, \ldots, s_i^{\ell_i}\}$. We write $[\![S_1, \ldots, S_n]\!]^k$, where $k \geq 0$, for the set of strings

$$\{\epsilon\} \cup \{s_1\$s_2\$\cdots\$s_n \mid \text{ either } s_i \in S_i \text{ or } s_i = \#s_i^1\$\cdots\$s_i^{\ell_i}\# \text{ and } |\{i \in \{1,\ldots,n\} \mid s_i \in S_i\}| \leq k\}.$$

Given a string $s_1\$\cdots\$s_n \in [\![S_1, \ldots, S_n]\!]^k \setminus \{\epsilon\}$, we define

$$\mathsf{unf}(s_i) \quad = \quad \begin{cases} \{s_i\} & \text{if } s_i \in S_i \\ S_i & \text{otherwise,} \end{cases}$$

for each $i \in \{1, \ldots, n\}$. The *unfolding* of $s \in [\![S_1, \ldots, S_n]\!]^k$, denoted $\mathsf{unfolding}(s)$, is defined as

$$\begin{cases} \mathsf{unf}(s_1) \times \cdots \times \mathsf{unf}(s_n) & \text{if } s = s_1\$\cdots\$s_n \\ \emptyset & \text{if } s = \epsilon. \end{cases}$$

Logspace Compactors. We are now ready to introduce logspace compactors. A *logspace 2-2-transducer* has two read-only two-way input tapes. Our logspace compactors are essentially logspace 2-2-transducers that output strings of the shape introduced above. We use the notion of *certificate function*, which is a logspace computable function $g : \{0,1\}^* \to \mathbb{N}$ such that $g(x) \in O(\log |x|)$.

Definition 4.1 (**Logspace Compactor**). A *logspace k-compactor*, where $k \geq 0$, is a logspace 2-2-transducer M such that, for some certificate function g, the following holds: for each $x \in \{0,1\}^*$, there are non-empty sets $S_1, \ldots, S_n \subseteq \{0,1\}^*$, for $n \geq 0$, such that, for each $c \in \{0,1\}^*$ with $|c| \leq g(x)$, $M(x,c) \in [\![S_1, \ldots, S_n]\!]^k$. ∎

In simple words, a logspace k-compactor accepts as input an instance $x \in \{0,1\}^*$, and a candidate certificate $c \in \{0,1\}^*$ of logarithmic size, and outputs ϵ, or a compact representation of $[S_1, \ldots, S_n]^{\sigma_c}$, where σ_c is the ℓ-selector for $S_1, \ldots, S_n$ determined by c.

The Λ-hierarchy. Given a logspace k-compactor M, let unfold_M be the function $\{0,1\}^* \to \mathbb{N}$ such that $\mathsf{unfold}_M(x)$ is the number

$$\left| \bigcup_{c \in \{0,1\}^*, |c| \leq g(x)} \mathsf{unfolding}(M(x,c)) \right|.$$

For each $k \geq 0$, we can then define the complexity class

$$\Lambda[k] \quad = \quad \{\mathsf{unfold}_M \mid M \text{ is a logspace } k\text{-compactor}\}.$$

The Λ-hierarchy is defined as the infinite union

$$\Lambda \quad = \quad \bigcup_{k \geq 0} \Lambda[k].$$

We proceed to establish some fundamental results concerning the Λ-hierarchy. The first one is crucial for our analysis:

Proposition 4.2. *For each $k \geq 0$, $\Lambda[k]$ is closed under many-one logspace reductions.*

The above proposition, shown by exploiting the fact that many-one logspace reductions can be composed, simply says that, if $f \leq_{\mathrm{m}}^{\log} g$ and $g \in \Lambda[k]$, then $f \in \Lambda[k]$. The next result confirms that Λ is a hierarchy of classes inside SpanL:

Theorem 4.3. $\Lambda[0] \subseteq \Lambda[1] \subseteq \cdots \subseteq \Lambda \subseteq$ SpanL*; furthermore, unless* L = NL, $\Lambda \subsetneq$ SpanL.

The fact that $\Lambda[i] \subseteq \Lambda[i+1]$, for $i \geq 0$, follows by definition. For showing that $\Lambda \subseteq$ SpanL, we essentially show that a logspace k-compactor, for $k \geq 0$, can be converted into a logspace NTT that follows the guess-check-expand paradigm. This should be already clear from the previous semi-formal discussion on guess-check-expand algorithms and logspace compactors. Finally, for showing that $\Lambda \subsetneq$ SpanL (unless L = NL), we first establish that, for every $f \in \Lambda$, its decision version $f_{>0}$ is in L; the latter uses the fact that the existence of a solution is witnessed by a small, logspace verifiable, certificate. Since every problem in NL is the decision version of some problem in SpanL, SpanL $\subseteq \Lambda$ implies L = NL.

The next result establishes that, apart from $\Lambda[0]$ and $\Lambda[1]$, the rest of the hierarchy consists of "hard" classes:

Theorem 4.4. *The following statements hold:*
(1) $\Lambda[1] \subseteq$ #L*; furthermore, unless* L = NL, $\Lambda[1] \subsetneq$ #L.
(2) $\mathrm{FP}^{\Lambda[2]} = \mathrm{FP}^{\#\mathrm{P}}$.

For showing that $\Lambda[1] \subseteq$ #L, for every function $f \in \Lambda[1]$, we provide a logspace NTM M_f such that, for each $x \in \{0,1\}^*$, $f(x) = \mathrm{accept}_{M_f}(x)$. The fact that the inclusion is strict (unless L = NL) is a consequence of (i) for each $f \in \Lambda[1]$, $f_{>0} \in$ L (actually, as discussed above, this holds for the whole hierarchy), and (ii) every problem in NL is the decision version of a problem in #L. For showing the second claim, i.e., $\mathrm{FP}^{\Lambda[2]} = \mathrm{FP}^{\#\mathrm{P}}$, it suffices to show $\mathrm{FP}^{\#\mathrm{P}} \subseteq \mathrm{FP}^{\Lambda[2]}$ (the other direction is trivial), which is equivalent to $\#\mathrm{P} \subseteq \mathrm{FP}^{\Lambda[2]}$. The latter boils down to showing that there exists $f \in \Lambda[2]$ that is $\leq_{\mathrm{T}}^{\mathrm{p}}$-complete for #P. Such a function is #Pos2DNF,

which computes the number of satisfying assignments of a positive 2DNF formula. Indeed, we can easily show that #Pos2DNF $\in \Lambda[2]$, while the #P-hardness is implicit in [12]; for a formal proof see [8].

The second statement of Theorem 4.4 has a couple of interesting implications, which confirm that beyond $\Lambda[1]$ we deal with "hard" complexity classes. More precisely, the fact that $\mathrm{FP}^{\Lambda[2]} = \mathrm{FP}^{\#\mathrm{P}}$, combined with the fact that the polynomial-hierarchy (PH) is included in $\mathrm{P}^{\#\mathrm{P}}$ (this is a consequence of Toda's powerful theorem [13]), we get that:

Corollary 4.5. *(1) If* $\Lambda[2] \subseteq \mathrm{FP}$*, then* $\mathrm{P} = \mathrm{NP}$.
(2) $\mathrm{PH} \subseteq \mathrm{P}^{\Lambda[2]} = \mathrm{P}^{\#\mathrm{P}}$.

Notice that claim (1) of Theorem 4.4, together with claim (1) of the above corollary, imply that the 1st level of the Λ-hierarchy is strictly contained in the 2nd level, unless $\mathrm{P} = \mathrm{NP}$. We can also show, assuming that there are infinitely many Mersenne primes, i.e., prime numbers of the form $2^n - 1$, for some integer n, that the 0th level is strictly contained in the 1st level. The above assumption is known as the Lenstra-Pomerance-Wagstaff conjecture. Therefore:

Proposition 4.6. *(1) Unless* $\mathrm{P} = \mathrm{NP}$, $\Lambda[1] \subsetneq \Lambda[2]$.
(2) Unless the Lenstra-Pomerance-Wagstaff conjecture does not hold, $\Lambda[0] \subsetneq \Lambda[1]$.

This brings us to the crucial question whether the same can be established (possibly under reasonable assumptions) for the other levels of the hierarchy. Unfortunately, we do not known whether the inclusions beyond $\Lambda[2]$ are strict. This essentially asks whether the Λ-hierarchy collapses to its kth level for some $k \geq 2$, i.e., whether $\Lambda = \Lambda[k]$. This is a difficult question that, despite our efforts, has remained open. We strongly believe that:

Conjecture 4.7. *The Λ-hierarchy does not collapse.*

An interesting issue is whether our new classes ensure the existence of efficient approximation schemes; this is studied in Section 6. But now the question is whether the Λ-hierarchy serves its purpose. This is the subject of the next section.

5 COMPLEXITY OF #CQA($\exists\mathbb{F}\mathbb{O}^+$)

We are now ready to analyze the complexity of #CQA($\exists\mathbb{F}\mathbb{O}^+$) under the lenses of many-one logspace reductions. We first study, in Section 5.1, the question whether a covering function for #CQA($\exists\mathbb{F}\mathbb{O}^+$) that allows to pinpoint the data complexity of the parameterized version of the problem exists. Then, in Section 5.2, we partially answer the question whether we can characterize the data complexity of #CQA($\exists\mathbb{F}\mathbb{O}^+$) via a class $C \subseteq$ SpanL.

Input: a database D, and a pair (Q', h)
Output: a string from $[\![B_1, \ldots, B_n]\!]^k$

if $h(Q') \nsubseteq D$ or $h(Q') \not\models \Sigma$ **then**
  halt;
$i := 1$;
while $i \leq n$ **do**
  if $B_i \cap h(Q') = \{R(\bar{t})\}$ and Σ has an R-key **then**
    output $R(\bar{t})$;
  else
    output $\#\alpha_i^1\$ \cdots \$\alpha_i^{\ell_i}\#$;
  if $i < n$ **then**
    output \$;
  $i := i + 1$;

Algorithm 2: The k-compactor $M_{Q,\Sigma}$.

5.1 #CQA($\exists\mathbb{F}\mathbb{O}^+$) and the Keywidth Function

Recall that the keywidth function (kw) maps pairs (Q, Σ), where $Q \in \exists\mathbb{F}\mathbb{O}^+$ and Σ is a set of primary keys, to $\mathbb{N}$, while

$$kw(Q, \Sigma) = |\{R(\bar{t}) \mid R(\bar{t}) \text{ occurs in } Q, \text{ and } \Sigma \text{ has an } R\text{-key}\}|.$$

It is clear that kw is a covering function for #CQA($\exists\mathbb{F}\mathbb{O}^+$). We focus on the parameterized version of #CQA($\exists\mathbb{F}\mathbb{O}^+$), denoted $\#\mathrm{CQA}_k^{kw}(\exists\mathbb{F}\mathbb{O}^+)$, which is defined in the same way as #CQA($\exists\mathbb{F}\mathbb{O}^+$) with the key difference that the input query Q and set Σ of primary keys are such that $kw(Q, \Sigma) = k$.

Theorem 5.1. $\#\mathrm{CQA}_k^{kw}(\exists\mathbb{F}\mathbb{O}^+)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\Lambda[k]$.

For brevity, let $\mathsf{F}_k^{kw} = \{\#\mathrm{CQA}(Q, \Sigma) \mid kw(Q, \Sigma) = k\}$. The above result boils down to showing that, for each $k \geq 0$:

(1) (**Membership**) For each $f \in \mathsf{F}_k^{kw}$, $f \in \Lambda[k]$.
(2) (**Hardness**) There exists $f \in \mathsf{F}_k^{kw}$ that is $\leq_{\mathrm{m}}^{\log}$-complete for $\Lambda[k]$.

The rest of this subsection is devoted to showing the above statements. In the sequel, fix an arbitrary $k \geq 0$.

Proof of Membership

Since an existential positive query can be converted into a UCQ, we can assume that the function $f \in \mathsf{F}_k^{kw}$ is of the form #CQA(Q, Σ), where $Q \in \mathbb{UCQ}$, and Σ is a set of primary keys. We need to show that #CQA(Q, Σ) is a function of the form unfold_M, where M is a logspace k-compactor. Actually, we simply need to convert the guess-check-expand NTT, depicted in Algorithm 1, into a check-compact transducer. It should be clear from our semi-formal discussion how this can be done, but, for the sake of completeness, we provide here the details and the formal definition.

Consider the 2-2-transducer $M_{Q,\Sigma}$, depicted in Algorithm 2 (notice the similarity with Algorithm 1), that accepts as input

a database D, and a candidate certificate (Q', h), where Q' is a disjunct of Q and $h : \mathsf{var}(Q') \to \mathsf{dom}(D)$. The database D is stored on the first tape, while the pair (Q', h) on the second tape. As usual, $B_1, \ldots, B_n$ is the sequence of the sets of $\mathsf{block}_\Sigma(D)$ induced by $\prec_{D,\Sigma}$. For brevity, let $B_i = \{\alpha_i^1, \ldots, \alpha_i^{\ell_i}\}$, for each $i \in \{1, \ldots, n\}$. Let us clarify that if the transducer halts without writing anything on the output tape, this means that the output tape contains the empty string ϵ. It can be shown that $M_{Q,\Sigma}$ uses only logarithmic-space; in fact, the argument is the same as for the logspace NTT depicted in Algorithm 1. It is also clear that $M_{Q,\Sigma}$ is a k-compactor. Notice that the latter is a consequence of the fact that the number of atoms $R(\bar{t})$ occurring in Q' such that Σ has an R-key is at most k, which in turn implies that indeed $M_{Q,\Sigma}(D, (Q', h)) \in [\![B_1, \ldots, B_n]\!]^k$. Finally, with C being the set of all candidate certificates, i.e., pairs of the form (Q', h), where Q' is a disjunct of Q, and $h : \mathsf{var}(Q') \to \mathsf{dom}(D)$, it is easy to see that the number of repairs that entail Q is

$$\left| \bigcup_{(Q',h) \in C} \mathsf{unfolding}\left(M_{Q,\Sigma}(D, (Q', h))\right) \right|,$$

and the claim follows.

Proof of Hardness

We proceed to show that there exists a query $Q \in \exists\mathbb{FO}^+$ (actually, a conjunctive query), and a set Σ of primary keys, with $kw(Q, \Sigma) = k$, such that, for every function $\lambda \in \Lambda[k]$, λ is many-one logspace reducible to #CQA(Q, Σ). The conjunctive query Q is defined as

$$\exists x_1 \exists y_1 \cdots \exists x_k \exists y_k \exists z \Big(\mathsf{Selector}(z, x_1, y_1, \ldots, x_k, y_k) \wedge \bigwedge_{i=1}^{k} \mathsf{Element}(x_i, y_i) \Big),$$

while the set Σ consists of the single key

$$\mathsf{key}(\mathsf{Element}) = \{1\}.$$

Roughly, the predicate Selector stores ℓ-selectors, with $\ell \leq k$, for the solution domains, while the predicate Element stores elements of the solution domains (e.g., Element(i, s) refers to the element s from the ith solution domain). It is clear that $kw(Q, \Sigma) = k$. We proceed to show that, for each function $\lambda \in \Lambda[k]$, $\lambda \leq_{\mathrm{m}}^{\log}$ #CQA(Q, Σ).

Fix $\lambda \in \Lambda[k]$. By definition, λ is a function of the form unfold_M, where M is a logspace k-compactor. Therefore, for some certificate function g, the following holds: for each $x \in \{0,1\}^*$, there are non-empty sets $S_1, \ldots, S_n \subseteq \{0,1\}^*$ such that, for each $c \in \{0,1\}^*$ with $|c| \leq g(x)$, $M(x, c) \in [\![S_1, \ldots, S_n]\!]^k$. Consider an arbitrary instance $x \in \{0,1\}^*$. We define the database D_x as the union of two databases: $D_{element}$ and $D_{selector}$. The first one collects all the elements of the solution domains that the compactor outputs:

$$\begin{aligned} &\{\mathsf{Element}(\star, \star)\} \cup \\ &\{\mathsf{Element}(i, s) \mid i \in \{1, \ldots, n\}, s \in S_i, \text{ and there exists} \\ &\qquad c \in \{0,1\}^* \text{ with } |c| \leq g(x) \text{ such that} \\ &\qquad M(x, c) = s_1\$ \cdots \$s_n \text{ with } s_i = s \text{ or } s_i = \# \cdots s \cdots \#\}. \end{aligned}$$

Let us clarify that Element$(\star, \star)$ is an auxiliary fact, which is needed for dealing with selectors that select less than k elements. The other database collects all the ℓ-selectors for the solution domains. For brevity, given a string $s_1\$ \cdots \$s_n \in [\![S_1, \ldots, S_n]\!]^k$, let $\mathsf{single}(s_1\$ \cdots \$s_n) = \{(i, s_i) \mid i \in \{1, \ldots, n\} \text{ and } s_i \in S_i\}$. Let also $C = \{c \in \{0,1\}^* \mid |c| \leq g(x), M(x, c) \neq \epsilon\}$. $D_{selector}$ follows:

$$\bigcup_{c \in C} \Big\{ \mathsf{Selector}(c, i_1, s_1, \ldots, i_\ell, s_\ell, \underbrace{\star, \ldots, \star}_{2(k-\ell)}) \mid \mathsf{single}(M(x, c)) = \{(i_j, s_j)\}_{j \in \{1, \ldots, \ell\}} \Big\}.$$

This completes the definition of D_x. We can show that

$$\lambda(x) = |\{D \in \mathsf{rep}(D_x, \Sigma) \mid D \models Q\}|.$$

This is done by exhibiting a bijection

$$h : \{D \in \mathsf{rep}(D_x, \Sigma) \mid D \models Q\} \to \bigcup_{\substack{c \in \{0,1\}^* \\ |c| \leq g(x)}} \mathsf{unfolding}(M(x, c));$$

$h(D') = s_1, \ldots, s_n$, where, for $i \in \{1, \ldots, n\}$, $\mathsf{Element}(i, s_i) \in D' \setminus \mathsf{Element}(\star, \star)$. We can also show, via simulation of the compactor M, that D_x can be constructed in logarithmic space. Thus, the above is a many-one logspace reduction.

5.2 Non-parameterized #CQA$(\exists\mathbb{FO}^+)$

We now focus our attention on #CQA$(\exists\mathbb{FO}^+)$. By exploiting Theorem 5.1, we can partially answer the question whether there is a class $C \subseteq$ SpanL, closed under many-one logspace reductions, such that #CQA$(\exists\mathbb{FO}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for C.

Theorem 5.2. *The following are equivalent:*

(1) There exists a complexity class $C \subseteq$ SpanL, closed under many-one logspace reductions, such that #CQA$(\exists\mathbb{FO}^+)$ *is $\leq_{\mathrm{m}}^{\log}$-complete for C in data complexity.*
(2) There exists an integer $k \geq 0$ such that #CQA$(\exists\mathbb{FO}^+)$ *is $\leq_{\mathrm{m}}^{\log}$-complete for $\Lambda[k]$ in data complexity.*
(3) The Λ-hierarchy collapses.

Theorem 5.2 essentially tells that either #CQA$(\exists\mathbb{FO}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\Lambda[k]$, for some $k \geq 0$, or there is no complexity class $C \subseteq$ SpanL, closed under logspace reductions, such that #CQA$(\exists\mathbb{FO}^+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for C. Thus, the

right complexity class that characterizes the data complexity of our problem, if it exists, is one of the levels of the Λ-hierarchy. Moreover, whether such a class exists boils down to the question whether the Λ-hierarchy collapses. Thus, the latter question begs for an answer, which will complete the picture concerning the data complexity of $\#\mathsf{CQA}(\exists\mathbb{FO}^+)$.

6 EFFICIENT APPROXIMATION SCHEMES

When a counting problem is computationally hard, it is natural to approximate it. In general, approximations to such problems can be computed via randomized algorithms with some probabilistic guarantees. Here we target fully polynomial-time randomized approximations schemes, FPRAS for short. As we shall see, for arbitrary first-order queries, our problem does not admit an FPRAS (under a reasonable assumption). On the other hand, for positive existential queries, we already know that our problem admits an FPRAS; this is implicit in [5]. However, by exploiting the fact that, for every $Q \in \exists\mathbb{FO}^+$ and set Σ of primary keys, $\#\mathsf{CQA}(Q, \Sigma)$ is in $\Lambda[kw(Q, \Sigma)]$, we can provide a conceptually simpler FPRAS. But let us first recall some basic notions.

Probability Spaces and Random Variables. A *probability space* is a pair $\mathsf{PS} = (\Omega, \pi)$, where Ω is a finite set, called *sample space*, and $\pi : \Omega \to [0, 1]$ is a function such that $\sum_{\omega\in\Omega} \pi(\omega) = 1$. A subset $E \subseteq \Omega$ is called an *event*. The probability of an event E, denoted $\mathsf{Pr}(E)$, is defined as $\sum_{\omega\in E} \pi(\omega)$. A *random variable* over PS is a function $X : \Omega \to \mathbb{Q}$. The *probability distribution* of X is a function π_X from the image of X to $[0, 1]$ such that $\pi_X(x) = \mathsf{Pr}(X = x)$, where $X = x$ denotes the event $\{\omega \in \Omega \mid X(\omega) = x\}$.

FPRAS. Consider a function $f : \{0, 1\}^* \to \mathbb{N}$. A *fully polynomial-time randomized approximation scheme* (FPRAS) for f is a randomized algorithm A that accepts as input $x \in \{0, 1\}^*$, and numbers $\epsilon > 0$ and $0 < \delta < 1$, runs in polynomial time in $|x|$, $1/\epsilon$ and $\log(1/\delta)$, and produces a random variable $\mathsf{A}(x, \epsilon, \delta)$ such that

$$\mathsf{Pr}(|\mathsf{A}(x, \epsilon, \delta) - f(x)| \;\leq\; \epsilon \cdot f(x)) \;\geq\; 1 - \delta.$$

Let us clarify that, assuming the random variable $\mathsf{A}(x, \epsilon, \delta)$ is over the sample space (Ω, π), $|\mathsf{A}(x, \epsilon, \delta) - f(x)| \leq \epsilon \cdot f(x)$ denotes the event $\{\omega \in \Omega \mid |\mathsf{A}(x, \epsilon, \delta)(\omega) - f(x)| \leq \epsilon \cdot f(x)\}$.

6.1 Approximation via FPRAS

As said above, for arbitrary first-order queries, our problem does not admit an FPRAS, under the assumption that RP $\neq$ NP. This is a consequence of the fact that $\#\mathsf{CQA}(\mathbb{FO})$ is $\leq^{\log}_{\mathrm{m}}$-complete for #P (Theorem 3.3). Recall that RP is the class of problems that are efficiently solvable by a randomized algorithm with a bounded one-sided error (i.e., the answer may mistakenly be "no") [3].

Input: a string $x \in \{0, 1\}^*$
Output: 0 or 1

for $i = 1$ to n **do**
  choose $s_i \in S_i$ with probability $\frac{1}{|S_i|}$;
if there exists $c \in \{0, 1\}^*$ with $|c| \leq g(x)$ such that $(s_1, \ldots, s_n) \in \mathsf{unfolding}(M(x, c))$ **then**
  return 1;
else
  return 0;

Algorithm 3: The randomized algorithm Sample.

Theorem 6.1. *Unless* RP = NP*, there exists a query* $Q \in \mathbb{FO}$ *and a set* Σ *of primary keys such that* $\#\mathsf{CQA}(Q, \Sigma)$ *does not admit an FPRAS.*

We now proceed to provide an FPRAS for the case of positive existential queries which, as mentioned above, is conceptually simpler than the one inherited from [5]. Actually, we are going to establish a more general result, which will immediately imply the desired FPRAS. Interestingly, we can show that every problem that falls in the Λ-hierarchy admits such a conceptually simple FPRAS:

Theorem 6.2. *For* $k \geq 0$*, every function in* $\Lambda[k]$ *admits an FPRAS.*

Proof. Fix $k \geq 0$, and consider a function $f \in \Lambda[k]$. By definition, f is a function of the form unfold_M, where M is a logspace k-compactor. Let g be the certificate function associated with M. Consider an arbitrary $x \in \{0, 1\}^*$, and let $S_1, \ldots, S_n \subseteq \{0, 1\}^*$ be the solution domains of M on input x. The FPRAS for f relies on a randomized algorithm, called Sample, which is depicted in Algorithm 3. We define U as the cartesian product $S_1 \times \cdots \times S_n$. We can show that:

Lemma 6.3. *The following statements hold:*

(1) $\mathsf{Sample}(x)$ *terminates after polynomially many steps.*
(2) $\mathsf{Pr}(\mathsf{Sample}(x) = 1) = \frac{f(x)}{|U|}$.

Sample is a polynomial-time randomized algorithm that, on input x, outputs a random variable X_x over (U, π), with π being the uniform distribution, that maps U to $\{0, 1\}$, and $\mathsf{Pr}(X_x = 1) = \frac{f(x)}{|U|}$. Consider now the random variable

$$X_x^{(t)} \;=\; \frac{|U|}{t} \cdot \sum_{i=1}^{t} X_x^i,$$

where $t > 0$, and $X_x^1, \ldots, X_x^t$ are independent random variables, which are essentially copies of X_x. Having $X_x^{(t)}$ in place, we define the randomized algorithm Apx_f, which accepts as

input x, and numbers $\epsilon > 0$ and $0 < \delta < 1$, and performs the following steps:

(1) $m := \max_{1 \le i \le n} \{|S_i|\}$;
(2) $t := \frac{(2+\epsilon)m^k}{\epsilon^2} \cdot \ln\left(\frac{2}{\delta}\right)$;
(3) return $X_x^{(t)}$.

It is easy to see that Apx_f runs in polynomial time in $|x|$, $1/\epsilon$ and $\ln(1/\delta)$. It remains to show that Apx_f is indeed an FPRAS for f. To this end, we exploit Chernoff's inequality. Since $X_x^{(t)}$ is the product of $|U|$ and the random mean of t Bernoulli random variables,

$$\Pr\left(\left|X_x^{(t)} - f(x)\right| \;\le\; \epsilon \cdot f(x)\right) \;\ge\; 1 - 2e^{-\frac{t\epsilon^2}{2+\epsilon} \cdot \frac{f(x)}{|U|}}.$$

By replacing t with $\frac{(2+\epsilon)m^k}{\epsilon^2} \cdot \ln\left(\frac{2}{\delta}\right)$, we obtain that

$$\Pr\left(\left|X_x^{(t)} - f(x)\right| \;\le\; \epsilon \cdot f(x)\right) \;\ge\; 1 - 2e^{-m^k \cdot \ln\left(\frac{2}{\delta}\right) \cdot \frac{f(x)}{|U|}}.$$

Observe that, if we could replace $\frac{f(x)}{|U|}$ with $\frac{1}{m^k}$, then

$$\Pr\left(\left|X_x^{(t)} - f(x)\right| \;\le\; \epsilon \cdot f(x)\right) \;\ge\; 1 - \delta,$$

which proves the claim. Thus, it remains to show that $\frac{f(x)}{|U|} \ge \frac{1}{m^k}$. Consider a small certificate $c \in \{0,1\}^*$ with $|c| \le g(x)$, and let $s = s_1\$\cdots\$s_n = M(x,c)$. Since M is a k-compactor, $|\mathsf{single}(s)| = \ell \le k$; recall that $\mathsf{single}(s) = \{(i, s_i) \mid s_i \in S_i\}$. We can assume, w.l.o.g., that $\mathsf{single}(s) = \{(i, s_i)\}_{1 \le i \le \ell}$. Hence,

$$|\mathsf{unfolding}(M(x,c))| \;=\; \prod_{i=\ell+1}^{n} |S_i|.$$

By definition of $\Lambda[k]$, $f(x) \ge |\mathsf{unfolding}(M(x,c))|$. Thus,

$$\frac{f(x)}{|U|} \;\ge\; \frac{\prod_{i=\ell+1}^{n} |S_i|}{\prod_{i=1}^{n} |S_i|} \;=\; \frac{1}{\prod_{i=1}^{\ell} |S_i|} \;\ge\; \frac{1}{\prod_{i=1}^{k} |S_i|}.$$

Recall that $m = \max_{1 \le i \le n} \{|S_i|\}$. Consequently,

$$\frac{f(x)}{|U|} \;\ge\; \frac{1}{\prod_{i=1}^{k} m} \;=\; \frac{1}{m^k},$$

and the claim follows. □

An immediate corollary of Theorem 6.2 is that:

Corollary 6.4. *For every query $Q \in \exists\mathbb{FO}^+$ and set Σ of primary keys, $\#\mathsf{CQA}(Q, \Sigma)$ admits an FPRAS.*

As already said, it is known that the above statement holds. In particular, in [5] it is shown that, for every positive existential Boolean query Q, the problem of computing the probability of Q over a disjoint-independent probabilistic database, denoted $\mathsf{DisjPDB}(Q)$, admits an FPRAS.[1] But, it is an easy task, given a query $Q \in \exists\mathbb{FO}^+$ and a set Σ of primary keys, to reduce $\#\mathsf{CQA}(Q, \Sigma)$ to $\mathsf{DisjPDB}(Q)$ via an approximation-preserving polynomial-time Turing reduction, which in turn implies that $\#\mathsf{CQA}(Q, \Sigma)$ admits an FPRAS. Therefore, the novelty here is not the statement itself, but the actual FPRAS, which is conceptually simpler than the one inherited from [5].

The simplicity of our FPRAS lies in the fact that we can sample from the natural sample space, i.e., the solution domains of the problem (see Algorithm 3). This is not the case for $\mathsf{DisjPDB}(Q)$, where the natural sample space consists of all the possible worlds of the probabilistic database. However, by using this sample space, in general, we need exponentially many samples in order to achieve the desired error and confidence guarantees. Thus, the sampling should be performed over a more complex sample space consisting of pairs of the form (W, h), where W is a possible world, and h is a homomorphism from the query to the database.

[1] In fact, the result is given for conjunctive queries, but it can be easily generalized to positive existential queries.

7 MORE ON THE Λ-HIERARCHY

In this last section, we provide more completeness results for the Λ-hierarchy, which justifies even further its meaningfulness. This analysis led to the definition of another natural complexity class that goes beyond the Λ-hierarchy, but is still inside SpanL.

7.1 More Complete Problems

We present two families of problems that have a problem that is complete for every level of the Λ-hierarchy. The first one is about counting satisfying assignments of Boolean formulas, while the second one is about counting forbidden colorings for hypergraphs.

Counting Satisfying Assignments. Let $X = \{x_1, \ldots, x_n\}$ be a (possibly empty) set of Boolean variables, and let $P = \{X_1, \ldots, X_n\}$ be a partition of X. A *P-assignment* for X is a function $\mu : X \to \{0,1\}$ such that, for each $i \in \{1, \ldots, n\}$, exactly one $x \in X_i$ is such that $\mu(x) = 1$. A positive kDNF formula over X is a Boolean formula of the form $\varphi = C_1 \vee \cdots \vee C_n$, where each C_i is a conjunction of $\ell \le k$ variables from X occurring positively in C_i, i.e., the variables occurring in φ form a subset of X. A P-assignment μ for X *satisfies* φ if $\mu(\varphi)$, i.e., the formula obtained after replacing each variable x with $\mu(x)$, evaluates to 1. Then:

PROBLEM :	#DisjPos$k$DNF
INPUT :	A set of variables X, a partition P of X, and a positive kDNF formula φ over X.
OUTPUT :	Number of P-assignments for X that satisfy φ.

The above problem is a generalization of the problem of counting the number of satisfying assignments of a positive kDNF formula, and is the counting variant of a probabilistic

extension of kDNF presented in [5]. It is possible to show that, for each $k \geq 0$:

Theorem 7.1. #DisjPos$k$DNF *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\Lambda[k]$.

Counting Forbidden Colorings. Consider a hypergraph $H = (V, E)$; H is called *k-uniform*, for $k \geq 0$, if, for each $e \in E$, $|e| = k$. Let $C = \{C_v\}_{v \in V}$ be sets of colors for each node $v \in V$. A *C-assignment* for a set $S \subseteq V$ is a function $\mu : S \to \bigcup_{v \in V} C_v$ such that $\mu(v) \in C_v$. Given a set $F = \{F_e\}_{e \in E}$, where F_e is a set of C-assignments for e, a *forbidden C-coloring for H w.r.t. F* is a C-assignment μ for V such that, for some $e \in E$ and $\nu \in F_e$, $\mu(v) = \nu(v)$, for each $v \in e$. Then:

PROBLEM :	#kForbColoring
INPUT :	A k-uniform hypergraph $H = (V, E)$, sets of colors $C = \{C_v\}_{v \in V}$, and sets of C-assignments $F = \{F_e\}_{e \in E}$.
OUTPUT :	Number of forbidden C-colorings for H w.r.t. F.

The above problem generalizes the problem of counting the number of non-list-colorings of a hypergraph; see, e.g., [7]. It is possible to show that, for each $k \geq 0$:

Theorem 7.2. #kForbColoring *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* $\Lambda[k]$.

7.2 Beyond the Λ-hierarchy

The above results consider the "bounded" version of the problems in question; #DisjPos$k$DNF bounds the number of variables in a clause, while #kForbColoring the number of nodes in a hyperedge. But, what about the complexity of their "unbounded" version, i.e., #DisjPosDNF and #ForbColoring? This question led to a new complexity class, dubbed SpanLL, which is of independent interest, that goes beyond the Λ-hierarchy, but is strictly contained in SpanL (unless L = NL).

The Complexity Class SpanLL. As the reader may have already suspected, this class is defined in the same way as $\Lambda[k]$, for some $k \geq 0$, with the key difference that an arbitrary logspace compactor, instead of a k-compactor, is used, which is able to choose an element from an unbounded number of solution domains. Formally, with $[\![S_1, \ldots, S_n]\!]$ (as usual, $S_1, \ldots, S_n \subseteq \{0,1\}^*$ are non-empty sets of strings) defined as the set of strings

$$\{\epsilon\} \cup \{s_1\$s_2\$\cdots\$s_n \mid \text{ either } s_i \in S_i \text{ or } s_i = \#s_i^1\$\cdots\$s_i^{\ell_i}\#\},$$

i.e., is the same as $[\![S_1, \ldots, S_n]\!]^k$ but without the condition that $|\{i \in \{1, \ldots, n\} \mid s_i \in S_i\}| \leq k$, a *logspace compactor* is defined in exactly the same way as a logspace k-compactor (see Definition 4.1) with the crucial difference that $M(x, c) \in [\![S_1, \ldots, S_n]\!]$. Then:

$$\text{SpanLL} \quad = \quad \{\text{unfold}_M \mid M \text{ is a logspace compactor}\}.$$

We can indeed show that SpanLL goes beyond the Λ-hierarchy, buy stays inside SpanL. In particular:

Theorem 7.3. $\Lambda \subseteq$ SpanLL $\subseteq$ SpanL*; furthermore, unless* L = NL*,* SpanLL $\subsetneq$ SpanL.

We can also show that:

Theorem 7.4. *Every function in* SpanLL *admits an FPRAS.*

Notice, however, that for devising an FPRAS for functions from SpanLL, we are forced to sample from a more complex sample space, similar in spirit to that for DisjPDB(Q); see the discussion at the end of Section 6. Sampling from the natural sample space, i.e., the solution domains of the problem, will lead to an exponential time algorithm. Observe that the FPRAS given in the previous section for functions from $\Lambda[k]$ becomes exponential if we do not bound k.

Back to our Problems. Coming back to the problems in question, we can show the following:

Theorem 7.5. #DisjPosDNF *and* #ForbColoring *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* SpanLL.

An immediate consequence of Theorem 7.4 and Theorem 7.5 is that #DisjPosDNF and #ForbColoring admit an FPRAS. Notice, however, that for the "bounded" version of the problems, one can use the simpler FPRAS where the sampling is performed over the natural sample space.

8 THE NEXT STEPS

Although we have provided a definite answer to our second question (Theorem 5.1), our first question is only partially answered (Theorem 5.2). Providing a firm answer to this question boils down to answering the question whether the Λ-hierarchy collapses. Our conjecture is that it does not collapse. Proving or disproving this conjecture is one of our main priorities, which will complete the picture concerning the data complexity of #CQA($\exists\mathbb{FO}^+$).

We have stressed a couple of times that the new FPRAS provided for each level of the Λ-hierarchy is conceptually simpler than the one inherited from the literature. The question is how this will perform in practice. The right way to answer this question is to implement this approximation scheme, and design a proper set of experiments, which we intend to do in the followup work.

Acknowledgements. We thank the anonymous referees for their useful feedback. This work was supported by the EPSRC grant EP/S003800/1 EQUID, and the EPSRC Programme Grant EP/M025268/ VADA.

REFERENCES

[1] Carme Àlvarez and Birgit Jenner. 1993. A Very Hard log-Space Counting Class. *Theor. Comput. Sci.* 107, 1 (1993), 3–30.

[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[3] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.
[4] Marco Calautti, Leonid Libkin, and Andreas Pieris. 2018. An Operational Approach to Consistent Query Answering. In *PODS*. 239–251.
[5] Nilesh N. Dalvi and Dan Suciu. 2007. Management of probabilistic data: foundations and challenges. In *PODS*. 1–12.
[6] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. 2005. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.* 340, 3 (2005), 496–513.
[7] Penny E. Haxell and Jacques Verstraëte. 2010. List Coloring Hypergraphs. *Electr. J. Comb.* 17, 1 (2010).
[8] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.* 79, 6 (2013), 958–983.
[9] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*. 155–164.
[10] Aris Pagourtzis and Stathis Zachos. 2006. The Complexity of Counting Functions with Easy Decision Version. In *MFCS*. 741–752.
[11] Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley.
[12] J Scott Provan and Michael O. Ball. 1983. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM J. Comput.* 12 (1983), 777–788.
[13] Seinosuke Toda. 1991. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.* 20, 5 (1991), 865–877.